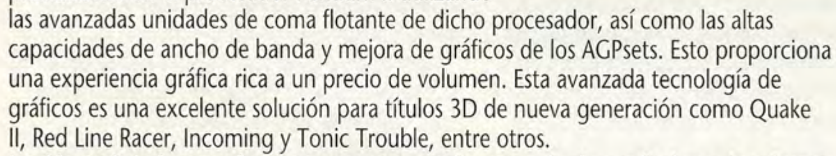


Intel anuncia el acelerador de gráficos 3d intel740

Intel Corporation anuncia su nuevo chip acelerador de gráficos, el Intel740. Optimizado para el procesador Pentium II, con AGPsets de Intel, el Intel740 es la fundación de los esfuerzos realizados por esta compañía para mejorar la experiencia visual de los usuarios de PCs. La arquitectura HyperPipelined 3D y la potente aceleración de vídeo y 2D introducen un alto nivel de realismo en el PC, para una informática de primera categoría, real como la vida misma. Al tiempo que la rápida adopción de la industria del interfaz AGP (Accelerated Graphics Port) conducida por Intel, junto con la avanzada tecnología de gráficos del Intel740, permiten una nueva generación de aplicaciones profesionales y de consumo para PC.



Según fuentes de Intel, el chip promete proporcionar altas prestaciones a los desarrolladores y usuarios finales. Esto, junto a la combinación del Intel740 y su sólida implementación en el API Direct3D estándar, significará una mayor potencia y flexibilidad para los desarrolladores de juegos y contenidos más interesantes para los usuarios finales.

El Intel740 está diseñado para equilibrar las prestaciones en el procesador Pentium II utilizando las avanzadas unidades de coma flotante de dicho procesador, así como las altas capacidades de ancho de banda y mejora de gráficos de los AGPsets. Esto proporciona una experiencia gráfica rica a un precio de volumen. Esta avanzada tecnología de gráficos es una excelente solución para títulos 3D de nueva generación como Quake II, Red Line Racer, Incoming y Tonic Trouble, entre otros.

Intel ha trabajado estrechamente con una amplia gama de desarrolladores de software para permitir contenidos de próxima generación para el procesador Pentium II con AGP y gráficos Intel740. Estas innovadoras aplicaciones 3D, como la visualización de datos, el CAD de consumo y los navegadores de Web en 3D, así como los títulos DVD de alta calidad y los nuevos juegos, que utilizan la potente aceleración del Intel740, contribuyen a la transformación de la informática en una experiencia visual incomparable para los usuarios de PCs corrientes.

La presentación de Windows 98 se prevé para el mes de mayo

Según fuentes norteamericanas es muy posible que Microsoft intente lanzar al mercado el ansiado sistema operativo el próximo día 1 de mayo coincidiendo con las nuevas plataformas de Intel de 350 a 400 Mhz.

Asimismo, Microsoft continuará con su política de bajo costes de actualización. De esta forma, todos los poseedores de Windows 95 podrán migrar al nuevo sistema operativo sin que sus bolsillos se resientan demasiado.



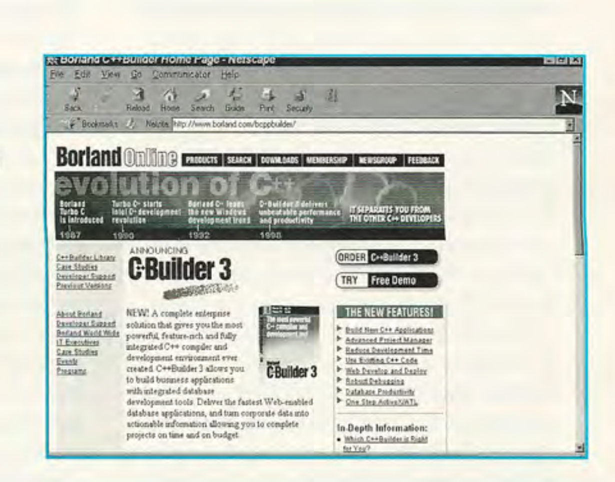
Sumario

- **3D Manía** 2
  Sumérgete de lleno en el mundo de la programación 3D de la mano de uno de los gurús españoles.
- **DIV** 5
  Sigue nuestro curso DIV. Es el mejor entorno para crear juegos de ordenador.
- **Taller 2D** 8
  Esta sección es la biblia del grafista. Palabra.
- **Diseño de Niveles** 11
  Todos los secretos para que construyas niveles a tu medida.
- **Taller Musical** 14
  Cuando la música se convierte en algo más que sonido.

Nuevo Borland C Builder 3

Borland ha anunciado el nuevo Borland C++Builder 3, la versión más reciente de su premiado sistema de desarrollo C++. Borland C++Builder 3 es un entorno de programación C++ avanzado para desarrolladores de sistemas y aplicaciones. La compañía americana anunció, además, las siguientes versiones del producto: Borland C++Builder Client Server Suite, Borland C++Builder Profesional y Borland C++Builder Estándar.

Borland C++Builder es un compilador de C++ ANSI/ISO de alto rendimiento que incluye más de 50 nuevas funciones, incluyendo funciones de programación C++ como un gestor de proyectos avanzado y tecnologías de depuración, acceso a bases de datos dimensionables integrado, desarrollo transparente en Internet, creación fácil de controles ActiveX/ATL en un solo paso, tecnologías Business y Code Insight para soporte de decisiones y asistencia en programación automatizada, y soporte de estándares del mercado, incluyendo Microsoft Foundation Classes 4.2 (MFC), Biblioteca de Objetos Windows (OWL), Biblioteca de Componentes Visual (VCL) y Standard Template Library 2.0 (STL).





En nuestro CD de portada incluimos el siguiente material:

- **Las fuentes de código de los ejemplos comentados en 3D manía.**
- **COOL EDIT: Editor de samples para generar efectos de sonido de calidad.**
- **Ejemplos del curso de DIV Games Studio.**

Autor: Jesús de Santos

Clipping [Parte I]

El recorte de polígonos o 'clipping' es un paso necesario en todo motor 3d. Este mes vamos a fijar los conceptos básicos de esta técnica para poder implementar, el mes que viene, un buen algoritmo de clipping 3d.

El recorte de polígonos, o más generalmente, **recorte** (a partir de ahora lo llamaremos recorte, a pesar de que es normal referirse a esta técnica por su nombre en inglés: **clipping**) es una parte básica en el proceso de pintado de una escena. Su misión es, como la propia palabra indica, recortar lo que vayamos a pintar en pantalla para que no salga fuera de los límites. Éste es un concepto que a los más principiantes puede parecer confuso, pero no hemos de olvidar que siempre que realizamos una escritura (a la pantalla directamente o a cualquier otro sitio) lo estamos haciendo en un rango de direcciones de memoria válidas y que fuera de esos límites no debemos nunca realizar accesos de escritura ni de lectura. Así, por ejemplo, en el caso más sencillo de un sprite que se mueve por pantalla habrá ocasiones en que el sprite está parcialmente dentro y fuera de los límites de la pantalla. Ante esta situación tenemos tres posibles soluciones a tomar:

- Pintar todo el sprite: no recomendado. Es posible que consigamos el cuelgue del sistema o que pintemos en zonas no deseadas con consecuencias imprevisibles.
- No pintar el sprite: en ciertas ocasiones esta solución puede ser buena. Es la más rápida de las tres, aún así, en la mayoría de los casos el resultado visual obtenido no es bueno, pues se nota claramente como los sprites desaparecen de golpe al llegar a los bordes de la pantalla. Sólo en el caso de que los sprites sean pequeños se puede obtener un resultado medianamente aceptable. Tampoco es recomendable esta opción.
- Recortar el sprite: sin duda la solución buena. Calculamos la porción del sprite que aparece en pantalla y visualizamos esa parte en pantalla. Visualmente se consigue que el sprite desaparezca suavemente de pantalla. En este caso, el cálculo de la porción de sprite visible es trivial, aunque no siempre va a ocurrir esto.

• **Clipping 2d**: Como su nombre indica, opera únicamente con coordenadas de pantalla (bidimensionales). El recorte del objeto a visualizar se realiza normalmente con la pantalla de visualización. Existen técnicas para recortar líneas, elipses, sprites, triángulos... Como trabajamos exclusivamente con coordenadas de pantalla, las operaciones a realizar son *sencillas y rápidas*. Pero esta rapidez se paga a costa de una serie de deficiencias que aparecen con este método. Generalmente, los objetos que recortaremos en pantalla vendrán de una escena 3d (a eso precisamente está dedicada esta sección de la revista). Por tanto, al recortar en 2d estamos perdiendo una dimensión que suele ser importante. Por ejemplo, supongamos un polígono que aparece en pantalla y que se introduce por la cámara. Este polígono necesita ser recortado en el plano z para quedarnos con la parte que está justo delante de la cámara (lógicamente, la parte que queda por detrás no nos interesa). Pero si realizamos el recorte en 2d no podemos operar con la tercera coordenada. Un problema similar aparece si queremos recortar los polígonos en profundidad (con el plano frontal y trasero de la cámara). Una posible solución a este problema es eliminar por completo el polígono que atraviese el plano z. Los resultados dependerán siempre del tipo de escena con la que estemos trabajando, pero no podemos aceptar este método como solución buena.

• **Clipping 3d**: Se realiza un recorte verdadero en tres dimensiones con los planos de visión de la cámara. El volumen de visualización de una cámara (siempre que trabajemos con una proyección cónica) tiene forma de pirámide truncada (lo que se denomina en inglés el *frustum view*). En total tenemos un volumen delimitado por 6 planos: derecho, izquierdo, superior, inferior, trasero y frontal. Todo objeto que entre en dicho volumen se proyectará en pantalla. El recorte selecciona la parte del objeto que entra dentro del volumen de visualización y que es la que se proyectará. Todos los problemas que se comentaban en el clipping 2d desaparecen en esta técnica. A cambio, tenemos que construir un algoritmo un poco más complejo.

CLIPPING 2D

Ya hemos explicado las bases del clipping 2d, así que vamos a ver cómo podríamos realizar el clipping de algo sencillo: una línea.
Podemos recortar una línea con diversos tipos de objetos, pero vamos a considerar un caso sencillo y que, por otro lado, es real pues es el único que vamos a utilizar. Vamos a considerar siempre recortes de elementos con rectángulos que, normalmente, tendrá las dimensiones de la pantalla. Por tanto, vamos a emplear siempre coordenadas de pantalla: la esquina superior izquierda de la pantalla es el origen, el eje horizontal se extiende positivamente hacia la derecha y el vertical hacia abajo.
Para recortar una línea sólo necesitamos saber la situación de los puntos inicial y final de la misma. No tenemos que clippear todos los puntos de la línea (esto es lo que se denomina '*coherencia*' y que es una técnica muy potente como veremos en el clipping 3d, aunque en este caso parezca trivial, que lo es).
El estado de cada uno de los dos extremos de la recta lo consideramos con respecto a cada una de las aristas de la pantalla. Existen cuatro aristas que definen la pantalla: Horizontal Superior, Horizontal Inferior, Vertical Izquierda, Vertical Derecha. Para cada arista, un punto puede tener dos estados: dentro o fuera de la arista.
El algoritmo que vamos a emplear es uno de los más eficientes y populares que existen para recortar líneas: se trata del algoritmo de Cohen-Sutherland. Cada uno de los puntos extremos de la recta recibe un código de estado, que nos informa de la posición de ese punto con respecto a los cuatro ejes que

LISTADO 1

```
DWORD get_code(float x, float y) {
   DWORD ret_code=CIN;
   if(y < TOP) ret_code |= CTOP;
   else if(y > BOTTOM) ret_code |= CBOTTOM;
   if(x < LEFT) ret_code |= CLEFT;
   else if(x > RIGHT) ret_code |= CRIGHT;
   return ret_code;
}
```

forman la ventana de recorte. Nosotros vamos a emplear la siguiente codificación en binario:

CIN	0000 ->	Punto dentro de pantalla
CTOP	0001 ->	Punto se sale por arriba
CBOTTOM	0010 ->	Punto se sale por abajo
CRIGHT	0100 ->	Punto se sale por la derecha
CLEFT	1000 ->	Punto se sale por la izquierda

En el listado número 1 tenemos una función que calcula el código correspondiente a un punto de pantalla de dimensiones (*left,right, top, bottom*): Una vez que tenemos el código correspondiente a los dos puntos, se tiene que dar uno de los tres siguientes casos:

1. Si los dos puntos entran en pantalla (ambos tienen el código asociado CIN) entonces ya hemos acabado con el recorte. Tenemos una recta que podemos pintar directamente sin tener que recortar.
2. Si realizamos un AND entre el código de cada punto y obtenemos un valor que no es cero, entonces tenemos el caso en el que los dos puntos se salen por una misma arista de la pantalla. En este caso, es claro que la recta está totalmente fuera de la pantalla y que no hay que pintarla. También, en este caso, hemos acabado con el recorte.
3. Si no se da ninguno de los dos casos anteriores entonces, al menos, uno de los dos puntos está fuera de alguna de las aristas: tomamos el punto y calculamos su intersección con la arista respecto a la cual está fuera. Obtenemos un nuevo punto del cual obtenemos su código de estado. Sustituimos el punto final de la recta que estamos considerando por el punto de intersección. Una vez hecho, esto empezamos todo el algoritmo desde el principio considerando la nueva recta, así hasta que acabemos en el punto 1 o 2.

En la Figura 1 tenemos los tres tipos básicos que se pueden dar.
En el listado 2 tenemos la implementación del algoritmo.
El algoritmo de Cohen-Sutherland se puede extender fácilmente a 3d. En este caso, en vez de calcular la posición de un punto con respecto a una arista lo haríamos con respecto a un plano. Ya vimos el mes pasado cómo medir distancias de puntos a planos. Existen otros muchos algoritmos para recortar líneas, el que hemos visto es particularmente útil para el caso en el que no se realizan muchos recortes. Esto es lo que suele ocurrir la mayoría de las ocasiones. En caso de que se realicen muchos recortes sobre una misma línea, se podría implementar un algoritmo que calculara la pendiente de la recta una sola vez (en el código de arriba la estamos calculando varias veces). Además, el algoritmo descrito es muy sencillo de portar a ensamblador y muy eficiente en máquinas que incorporen instrucciones máquina para realizar OR y AND.
Como siguiente ejemplo de clipping 2d vamos a realizar algo más productivo: recortar un **polígono**. Al igual que en el caso de la línea vamos a considerar algunas simplificaciones: una ventana rectangular como elemento de recorte y un polígono de *n* lados, que puede ser cóncavo o convexo.
El algoritmo que sigue a continuación es el de Sutherland-Hodgman. La idea principal de este algoritmo es descomponer el problema en partes más sencillas y que sean más fáciles de resolver que el caso inicial. Lo que vamos a hacer es recortar el polígono con cada una de las aristas, una a una. Primero recortamos con la superior y el polígono resultado que obtenemos lo recortamos con la arista inferior, así, sucesivamente, hasta que obtenemos un polígono final, que está recortado con respecto al total de la ventana de recorte. Este método es opuesto al que hemos empleado para líneas, donde recortábamos considerando simultáneamente todas las aristas de recorte.
Gráficamente, podemos ver en la figura 2 cómo realizamos el clipping parcial para cada una de las aristas de recorte.
Por tanto, tenemos que centrarnos en el recorte de un polígono con respecto a una arista. En este caso, estamos ante una situación similar al ejemplo que vimos anteriormente de la línea. Tenemos que ir operando con cada uno de los lados del polígono. Nos encontramos con las siguientes situaciones para cada uno de los lados del polígono:

1. Que los dos vértices del lado del polígono estén dentro de la arista. En este caso, la arista es completamente válida y se incluirá en el polígono resultado.
2. Que los dos vértices estén ambos fuera de la arista. La arista no es válida y no debe aparecer en el polígono recortado.
3. Un punto fuera y otro dentro. Tenemos que calcular el punto de intersección del lado con la arista y considerar como válido el segmento que queda dentro de la arista.

En una primera aproximación al algoritmo observamos que es más sencillo recortar el polígono, considerándolo como una sucesión de puntos más que como una sucesión de aristas. En seguida vamos a ver por qué. El algoritmo queda del siguiente modo:

1. Para cada uno de los vértices del polígono a recortar con una arista:
   1.1 Chequeamos su estado con respecto a la arista (Fuera o Dentro).
   1.2 Si tiene un estado distinto al del último vértice chequeado entonces existe un lado que intersecciona con la arista. Calculamos el punto de intersección de la recta determinada por el vértice actual y el anterior con la arista de recorte y añadimos el vértice al polígono de salida.
   1.3 Si el vértice actual está dentro de la arista añadirlo al polígono de salida.

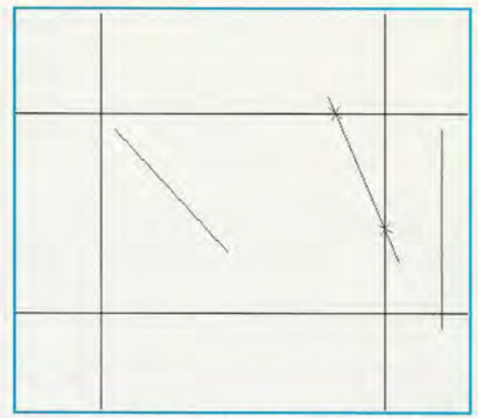
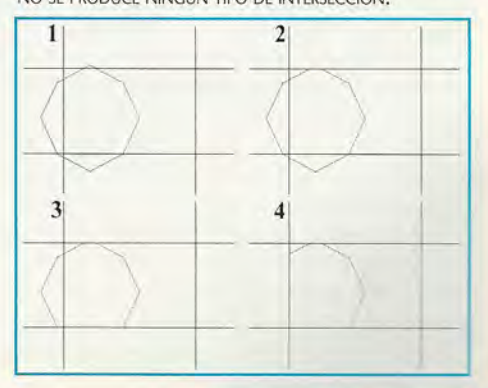


Figura 1. Diferentes posibilidades de recorte de una linea con respecto a una ventana de recorte.

Una vez finalizado el algoritmo tenemos una lista de vértices que definen el polígono recortado con respecto a una arista de recorte.
Si repetimos el proceso para cada una de las aristas que definen la ventana de visualización, obtenemos un polígono final que entra completamente en la ventana de visualización.
Destacar que hemos de tomar como parámetro de entrada para cada recorte la salida del último recorte; menos en el primer caso que, lógicamente, debe tomar el polígono inicial.
Una posible implementación en C ++ del algoritmo descrito es la siguiente:
Conviene destacar algunas cosas que nos serán de gran utilidad en el artículo del mes que viene. Si la ventana de recorte que estamos considerando es la pantalla, es muy recomendable chequear cada uno de los vértices de salida con los límites de la pantalla. En teoría, todos los puntos del polígono entran

Figura 2. El algoritmo descrito va recortando el poligono con cada una de las aristas de recorte. Primero con la de arriba, luego la de abajo y, finalmente, la de la izquierda. Con la de la derecha no se produce ningun tipo de interseccion.



LISTADO 2

```
void draw_clipline(float x0, float y0, float x1, float y1) {
  BOOLEAN clipping=TRUE;
  BOOLEAN draw=TRUE;
  DWORD code,code1, code2;
  float xt,yt;
// Calculo de los códigos iniciales para los dos extremos de la
// recta.
  code1=get_code(x0,y0);
  code2=get_code(x1,y1);
// Tenemos que repetir este bucle hasta que los dos puntos queden
// dentro de la ventana.
  while(clipping) {
// 1er Caso Trivial : Los dos puntos dentro
    if(code1 == CIN && code2 == CIN) clipping=FALSE;
// 2º Caso Trivial : Los dos puntos fuera
    else if(code1 & code2) {
      clipping=FALSE;
      draw=FALSE;
    }
// Caso no Trivial : al menos uno de los puntos esta fuera de
// pantalla
    else {
// Tomamos uno de los códigos que no sea CIN
      if(code1 != CIN) code=code1;
      else code=code2;
// Punto se sale por arriba
      if(code & CTOP) {
        xt = x0 + (x1 - x0)*(TOP - y0)/(y1 - y0);
        yt = TOP;
      }
// Punto se sale por abajo
      else if(code & CBOTTOM) {
        xt = x0 + (x1 - x0)*(BOTTOM - y0)/(y1 - y0);
        yt = BOTTOM;
      }
// Punto se sale por la izquierda
      else if(code & CLEFT) {
        xt=LEFT;
        yt=y0 + (y1 - y0)*(LEFT - x0)/(x1 - x0);
      }
// Punto se sale por la derecha
      else if(code & CRIGHT) {
        xt=RIGHT;
        yt=y0 + (y1 - y0)*(RIGHT - x0)/(x1 - x0);
      }
// Guardar el nuevo vértice creado en la nueva posición
      if(code1 != CIN) {
        x0=xt;
        y0=yt;
        code1=get_code(x0,y0);
      }
      else {
        x1=xt;
        y1=yt;
        code2=get_code(x1,y1);
      }
    }
  }
  if(draw) draw_line(x0,y0,x1,y1);
}
```

en pantalla, pero sólo en teoría. Debido a imprecisiones de la coma flotante, podemos obtener valores ligeramente fuera de la pantalla (un píxel como mucho). Escribir fuera de pantalla puede tener consecuencias imprevisibles como ya se dijo al principio del artículo. En los recortes de dos dimensiones la probabilidad de que ocurra esto es menor que en los tridimensionales. Ya lo veremos el mes que viene. El chequeo es simple:

```
if( xp < view.left ) xp = view.left;
else if( xp > view.right - 1 ) xp = view.right - 1;

if(yp < view.top ) yp = view.top;
else if( yp > view.bottom -1 ) yp = view.bottom -1;
```

LISTADO 3

```
POLY2D clip_poly_edge(POLY2D *p, EDGE
*edge) {
  POLY2D clipped;
  DWORD i,k;
  BOOLEAN last,cur;
  DWORD new_vrt=0;
  k=p->no_vrt-1;
  last=in(&p->vrt[k],edge);
// La función in() calcula el
// estado del vértice con
// respecto a edge
  for(i=0;i<p->no_vrt;i++) {
    cur=in(&p->vrt[i],edge);
    if(cur!=last) clipped.vrt[new_vrt++]=
interpolate(&p->vrt[k],&p->vrt[i],edge);
  if(cur) clipped.vrt[new_vrt++]=p->vrt[i];
  last=cur;
  k=i;
  }
  clipped.no_vrt=new_vrt;
  return clipped;

}
POLY2D clip_poly(POLY2D *p, WINDOW *rect) {
  POLY2D temp_poly;
  temp_poly = clip_poly_edge(p,&rect-
>side[0]);
  temp_poly =
clip_poly_edge(&temp_poly,&rect->side[1]);
  temp_poly =
clip_poly_edge(&temp_poly,&rect->side[2]);
  temp_poly =
clip_poly_edge(&temp_poly,&rect->side[3]);
  return temp_poly;
}
```

En el proceso de recorte podemos realizar una modificación para optimizar en velocidad. Si en vez de pasar a la función de recorte una lista de aristas, pasamos una lista de punteros a aristas podemos copiar del polígono de entrada al polígono de salida muy rápidamente, pues sólo movemos punteros. Generalmente, esto suele ser útil cuando cada arista tiene información adicional además de las posiciones (x,y), como puede ser la z, iluminación (r,g,b), posición de las texturas ... Nosotros aplicaremos esta técnica cuando veamos el recorte tridimensional.

Jesús de Santos García
icg_ent@hotmail.com

Autor: Antonio Marchal

La técnica Filmation

Tal vez a mucha gente el nombre de *Ultimate (Play the Game)* no le diga nada, sin embargo a otros les traerá recuerdos de su juventud. ¡Qué tiempos aquellos! Este nombre es el de un grupo de programación de la época del Spectrum, que utilizó por primera vez una técnica nueva de programación a la hora de representar un videojuego.

Este grupo de programación, en su momento, realizó programas que triunfaron dentro del mundo del Spectrum. Hoy en día, para quien conozca el mundo de las consolas, este grupo se ha reconvertido en Rare Soft, los creadores del Donkey Kong de Super Nintendo. Pero volvamos a la época del Spectrum; entre los muchos juegos que crearon, resaltaremos Knightlore, que no sólo sobresale por su calidad sino por la utilización de la técnica Filmation. El argumento consistía en un hombre que se convertía en hombre lobo y debía evitar ese maleficio; para ello, dentro de un castillo tenía que conseguir elementos de una pócima. A éste le siguió Alien 8, en el que la misión cambiaba pero la técnica utilizada era la misma. Después de un periodo de tiempo aparecieron otros dos juegos, pero con la técnica mejorada; estos eran NigthShade y Gunfright. La mejora consistía en que, mientras en las primeras versiones la acción se realizaba dentro de una habitación, que iba cambiando según íbamos avanzando, en estos dos últimos juegos, el protagonista se situaba en el centro de la pantalla y el decorado era lo que se movía. Con esta última técnica también existía un sistema que hacía que se ocultaran las paredes si no permitían que viéramos la acción del juego. Actualmente, juegos como Diablo, hacen uso de esta técnica, pero mucho más mejorada, ya que utiliza luces, sombras, etc... pero las bases de funcionamiento, en principio, son las mismas.

LA FORMULA

Como para casi toda programación en 3D es necesario una fórmula matemática que convierta las coordenadas 3D del objetos a 2D de pantalla, consiguiendo, así, tener únicamente dos coordenadas que es donde posicionaremos el gráfico. Primeramente explicaremos la posición de las coordenadas. Es decir, si en las 2D de pantalla se podría indicar que se usan dos coordenadas *"x"* e *"y"*, siendo *"x"* la que indica la horizontal, yendo de izquierda a derecha, e *"y"* las coordenadas verticales, que van de arriba a abajo. En el caso de la técnica Filmation, usaremos 3 coordenadas que formaran una Y invertida. A estas tres coordenadas las llamaremos *"ax"*, *"az"* y *"ay"*, y posicionaremos la última en el palo superior de la Y invertida, la *"ax"* en el palo inferior de la izquierda, y *"az"* en el que queda, que es el inferior de la derecha. Todas las coordenadas partirán desde donde se encuentran las tres coordenadas y avanzarán en la dirección que indica los diferentes de la Y invertida.
Pero como hemos comentado al principio tenemos que convertir estas coordenadas *"ax"*, *"az"* y *"ay"*, a las de la pantalla *"x"* e *"y"*. Para conseguirlo usaremos la siguiente formula:

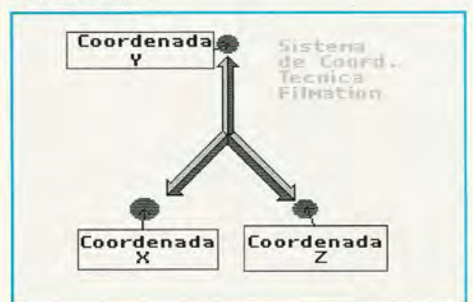
*x=ix+((ax-az)*2)*

*y=iy+(ax+az)-(2*ay)*

Las variables *"ix"* e *"iy"* son las coordenadas de inicio en la impresión. Es decir, si por ejemplo queremos que la coordenada 0,0,0 de nuestro sistema Filtmation aparezca en las coordenadas 100,160 de la pantalla, debemos usar estos valores como las coordenadas iniciales. Normalmente se usa el centro de la pantalla, si se va a realizar un scroll, mientras que si es únicamente una habitación, se utiliza la parte superior central de la pantalla, como coordenadas iniciales..
Asimismo podemos prescindir de la altura, si lo nuestro es el mundo plano; en ese caso las fórmulas se simplificarían, quedando de la siguiente manera:

*x=ix+((ax-az)*2)*
y=iy+(ax+az)

EL SISTEMA DE COORDENADAS EN PERSPECTIVA AXONOMETRICA.





Esto evita muchos problemas, como más adelante comprobaremos, pero le quita realismo a la acción del juego. La elección del uso de la altura, depende de las necesidades de cada uno.

FILMATION Y DIV, UNA PAREJA IDEAL

El implementar la técnica Filmation dentro del lenguaje DIV no crea muchos problemas. Al contrario, debido al modo de funcionamiento del lenguaje de programación de DIV, se evitan muchas de estas complicaciones. Pero, ¿cuáles son estos problemas? Que pasa, ¿qué no es usar una formulita y ya está? Pues no, ya que en todo sistema de gráficos en 3D, normalmente entra en juego un buen sistema de ordenación. Esto es debido, porque al contrario que sucede en la 2D que todo esta o arriba-abajo o a la izquierda-derecha del objeto, en los sistemas 3D, los gráficos también se sitúan por delante-detrás del objeto, lo que nos trae el problema de imprimirlos en el orden adecuado, para que todo quede en su posición correcta. El sistema Filmation no va a ser menos, por eso necesita una ordenación, situándose por detrás todo objeto cuya coordenada *"ax"*, *"az"* o *"ay"* esté por detrás de las que estemos comprobando en ese momento.
Como en todo problema, existen muchas formas de solucionarlo; hay que señalar que una de las soluciones que veremos sólo es posible en el lenguaje de DIV, debido a sus características, por ello si quisiéramos implementar este método con otro sistema, deberíamos programar bastantes líneas de código y, sin embargo, con DIV todo este proceso es automático. Por otro lado, existen muchas maneras de usar las fórmulas, pero una de las más convenientes es declarar *"ax"*, *"az"* y *"ay"*

como variable locales; de este modo conseguiremos que todos los procesos tengan tres nuevas variables que serán muy afines para nuestros propósitos. Luego, antes del FRAME de cada proceso, deberemos incluir la fórmula de cálculo de coordenadas y nuestro objeto tendrá coordenadas en axonométrica que serán las creadas, y que manipularemos a nuestro antojo.

"Z" DE ZORRO

Una de las variables predefinidas que nos brinda el entorno DIV Games Studio es *"z"*, que es local, con lo que cada proceso que creemos poseerá esta variable. Y ¿para qué sirve? Con ella controlaremos si un gráfico se imprime por encima o por debajo de otro. Seguro que habréis adivinado el primer truco de los que vamos a describir a continuación y que tiene que ver con la ordenación de los objetos. Éste se basa en usar las capacidades de la variable *"z"* para ordenar los objetos, asignando valores a cada uno de los procesos dependiendo de si debe aparece por delante o por detrás del objeto. La manera más sencilla de hacerlo es asignar el valor *"y"* de manera negativa, con la instrucción *z=-y*. Pero esto presenta algún problema. El primero de ellos es que la coordenada, que debemos tener en cuenta, debe estar bien colocada; esto lo conseguiremos asignando el primer punto de control a la posición correcta. Muchos de los objetos deberán tener el punto de control en la base, otros en el centro, etc...
Este método se puede completar usando distintos rangos de *"z"*, dependiendo del plano del proceso. Por ejemplo, se podrían asignar las *"z"* del 0 al -20 para los suelos, de las -20 a la -40 para los personajes de suelo, de la -40 a la -60 para los personajes que vuelen, y de la -60 en adelante para las nubes. Con esto se logra otra ordenación por campos que facilitará el trabajo de conseguir un cierto realismo en la visualización de nuestro videojuego.
El otro método es el que se utilizaba comúnmente en programación. Como todo existen varias maneras de hacerlo, pero la idea en que se basa es la misma: se empieza imprimiendo el gráfico más alejado y, después, se van imprimiendo los gráficos en orden uno encima de otro, dependiendo de la lejanía. Una manera de conseguirlo es pintar primero el suelo, para más tarde ordenar los objetos e imprimirlos en el orden correcto. Para ello deberemos usar un sistema de ordenación, como los sistemas de burbujas, de árbol, de fusión, etc. Para los que quieran conocer algún sistema de ordenación vamos a explicar uno de ellos que, aunque no sea muy rápido, pues existen decenas de métodos que superan a éste en todas sus características, si es práctico ya que nos solucionará el problema. Se coge el primer valor y se le compara con el segundo, con el tercero, con el cuarto,..., hasta que se encuentre un valor mayor, y se comprueba si éste es el del final; si es así se deja en esa posición, pasando el que era al principio el segundo a ser el primero, el que era el tercero pasa a ser el segundo, y así sucesivamente, hasta la posición donde tenemos que insertar el valor de lo que era la primera posición. Si se da el caso de que el valor que es mayor de lo que era el primero no es el final del fichero, se coge ese valor como si fuera el primero, se deja el primero en su posición y se empieza a comparar con este nuevo valor hasta que, siguiendo este último proceso, lleguemos al final del fichero. Todo este procedimiento se repite hasta que el primer valor sea menor que el segundo.

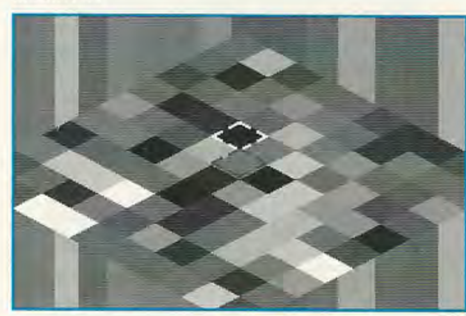
Hoy en día, juegos como Diablo hacen uso de esta técnica

Por si el sistema de ordenación no ha quedado claro, explicaremos otra manera de conseguir la técnica Filmation de manera artesanal, y que no necesita ningún tipo de ordenación. Como hemos explicado antes, para evitar la ordenación podemos imprimir el suelo antes que los objetos, comenzando por el más alejado y acabando por el más cercano. Esto es debido a que, normalmente, se tienen los datos de los gráficos en una tabla, cuyo elemento 0 es el que esta más alejado, mientras que el último elemento es el más cercano; ningún elemento de la tabla cuyo valor sea menor que otro se imprimirá después de éste, pues para evitar la ordenación de objetos, lo que haremos será comprobar en cada trozo del suelo si hay algún gráfico encima, y lo imprimiremos también en ese momento, de esta manera, todo quedará ordenado, de una sola pasada.

TILEAR O NO TILEAR...

Como se ha comentado antes, normalmente los gráficos del suelo suelen estar en una tabla. Esto ocurre cuando se usa la técnica, que ya comentamos en el artículo de scroll, y a la que denominamos *Tilear*. Para los que no hayan leído dicho artículo comentar que *tilear* es un sistema con el que construimos gráficos a base de bloques, es decir, si queremos una casa, haremos bloques, que serán los ladrillos, trozos de ventana de puerta, de tejado, etc.
Con la técnica Filmation se puede dar el caso de que tengamos que usar el *tileado*, si bien no se recomienda para objetos móviles, pero para otros elementos, como suelos y objetos estáticos, si queremos ganar en velocidad y necesitamos usar un sistema de bloques para construir los mapas, lo mejor que podemos hacer es construir un mapeador, que no es otra cosa que un programa que permite construir mapas a base de bloques. Estos bloques, que en realidad serán un número, que puede ser o no el número de orden dentro del gráfico en el fpg, formarán una tabla que guardaremos en disco para, desde el programa principal, cargar esta tabla y construir un gráfico mayor, usando la función *map_put()*. Este gráfico luego lo utilizaremos como un scroll cuyas variables serán *"ax0"*, *"az0"* y *"ay0"*, y que deberemos convertir a las usadas en la variable scroll, es decir, *"x0"* e *"y0"*, usando la fórmula descrita.

IMAGEN 1.



Funciones nombradas en el artículo

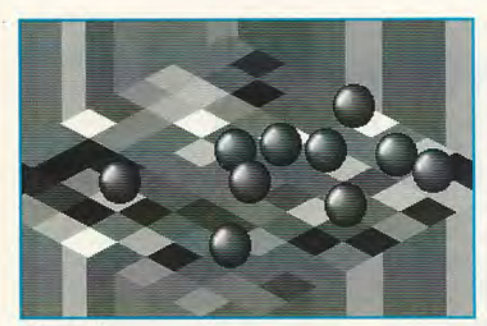
fget_dist(x1,y1,x2,y2)

Si queremos sustituir esta funcion lo podemos hacer usando para ello el teorema de Pitagoras, tal como se explica en articulo. Los parametros son las coordenadas de dos puntos, pero si usamos la tecnica Filmation, tendremos tres coordenadas. Si prescindimos de la altura podremos usar esta funcion, para calcular distancia entre objetos, si no deberemos usar el metodo descrito en el articulo.

Funciones nombradas en el artículo

map_put() y start_scroll()

Estas funciones ya se vieron en otros numeros de Game Over, si se quieren ver mas informacion tambien la podemos conseguir desde la documentacion de DIV. Pero las explicaremos por encima, la primera permite imprimir un grafico en otro, como parametros, necesitos los graficos, y las coordenadas de origen y destino. La otra funcion la usaremos para hacer scroll, y conjuntamente con la anterior para hacer mapas tileados. Como parametros de esta segunda funcion, deberemos indicar el numero de scroll, los graficos a usar como planos del mismo, la region donde aparecera, etc.


IMAGEN 2.

Funciones nombradas en el artículo

collision(id)

Si hacemos uso de la tecnica Filmation, y conseguimos mostrar un mundo 3D, el uso de esta funcion, como sucedia en los modos 7, ya no tiene sentido. Por eso debemos sustituirla con otras tecnicas de programacion. Cuando si se puede utilizar, la funcion collision(), usa como parametro el codigo identificador del proceso con el cual queremos detectar la colision, y se llama desde el otro proceso. Devolviendo la funcion un valor verdadero en el caso de que los dos procesos colisionen.

LAS COLISIONES

Otro de los problemas que nos encontramos al usar la técnica Filmation es la de la colisiones, ya que ésta deben ser detectadas aparte, como ocurría con los modos 7. Esto ocurre porque las verdaderas coordenadas del objetos son *"ax"*, *"az"* y *"ay"*, y no las habituales *"x"* e *"y"* que son sobre las que detecta la colisión la función *collision()*.El método usado para detectar si dos objetos chocan en 2D, aparte del uso de máscara, para la detección al píxel es muy parecido al de 3D y se basa en el cálculo de distancias. Si queremos que nuestro objeto tenga forma de cubo, con ancho, largo y alto, lo mejor es usar restas; este método es el más sencillo, por la fórmula usada, ya que la otra, la utilizada para hallar una forma circular, se basa en el sistema de Pitágoras, lo que implica el uso de raíces cuadradas.
Pero veamos el primer método con un ejemplo. Imaginemos dos procesos, *"pp1"* y *"pp2"*, a los que, además de declarar las variables locales *"ax"*, *"az"* y *"ay"*, tendremos otras tres variables que serán el ancho, el largo y el alto, y que designarán la medidas de los gráficos en cada coordenada, respectivamente. Para detectar si estos dos procesos colisionan deberemos saber si el espacio ocupado por un gráfico, siempre en 3 dimensiones, está dentro del otro. Para ello, usaremos una condición IF donde comprobaremos si el valor absoluto de la restas de las coordenadas es menor que un valor, de la siguiente manera:

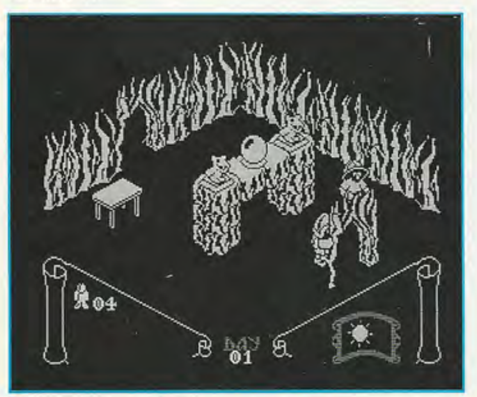
```
abs(pp1.ax-pp2.ax)>ancho AND
abs(pp1.az-pp2.az)>largo AND
abs(pp1.ay-pp2.ay)>alto
```

Pero este método da una forma cúbica a la colisión, por ello, si lo que queremos es detectar una colisión esférica o, mejor dicho, elíptica-esférica (es decir, una especie de esfera pero con distintos radios), sabiendo que para hallar el radio de un objeto, si tenemos el ancho y el alto, usando el teorema de Pitágoras, el radio sustituirá en dicho teorema a la hipotenusa. Para usar esto en tres dimensiones hallaremos un radio, usando dos de las coordenadas, y el resultado lo utilizaremos como cateto en la siguiente fórmula, utilizando también, como el otro cateto, a la otra coordenada que queda sin usar. Todo esto simplificando queda en que la hipotenusa es igual a la raíz cuadrada de la suma de las tres coordenadas al cuadrado, siendo igual que el teorema tradicional, cambiando únicamente el número de catetos. Esta fórmula nos dará un valor parecido a la función *fget_dist()* de DIV, pero esta vez en 3D.

LOS EJEMPLOS

Esta vez se incluyen tres ejemplos dentro del CD; el primero de ellos es un *tileador,* usando el sistema Filmation. Cambiaremos de loseta seleccionada con los cursores y modificaremos el color de la misma con el espacio. Aquí vemos cómo se utiliza la fórmula. En el siguiente ejemplo creamos un mapa al azar y ponemos gráficos estáticos encima. Aquí se puede ver el uso de la variable predefinida *"z"*, y comprobar cómo los objetos se superponen correctamente. Por último, en el otro ejemplo que resta, se demuestra cómo detectar colisiones y se aprecia cómo se mueven los objetos; asimismo, se hace uso de todas las técnicas, quedando un poco aparte el tema de las alturas, del cual no se ha puesto ningún ejemplo porque no implica ninguna dificultad en su uso. En este último ejemplo, el movimiento del cursor varía: en pantalla lleva la misma dirección que los cursores, mientras que dentro del entorno 3D se hace en diagonal.

IMAGEN 4.



HASTA LA VISTA, BABY

Bueno, este mes, el tema tenía mucho que ver con la programación. Y lo que en principio era una carencia del lenguaje, tras un estudio más detallado se puede comprobar que al usar este lenguaje a la hora de aplicar la técnica Filmation, el lenguaje tiene ciertas afinidades que le otorgan bastante atractivo a la hora de elegir el uso de esta técnica. Siempre tendremos un entorno 3D al usar este sistema, aunque sea desde un punto de vista bastante estático, se pueden realizar una visualización de mundos virtuales bastantes completas usando este método. Y siempre podremos programar varias vistas, cambiando las coordenadas de lectura, la *"ax"* por la *"az"*, y así poder ver objetos que, desde la otra perspectiva, pudieran queda ocultos.

ULTIMA HORA

Para despedirnos queremos recordar a aquellos que no la hayan visitado todavía que pueden visitar la página oficial de DIVGAMES STUDIO en Internet. Como se puede ver en la documentación que viene en el producto, podemos encontrar esta página en *http://www.divgames.com,* donde encontraréis información sobre el producto, conexiones, secciones divulgativas, conexiones a otros lugares y todo ese material que nos podemos encontrar en la red de redes. Si ha quedado alguna duda o, simplemente, os queréis poner en contacto con el autor de este artículo, podéis mandar un e-mail a *tizo@100mbps.es.* Nada más, esperamos que os DIVirtáis programando.

Funciones nombradas en el artículo

sqrt(valor)

Esta funcion devuelve la raiz cuadrada de un valor, y nos sera de gran utilidad para calcular distancias entre objetos. Cosa que debemos hacer para detectar colosiones entre objetos, ya que la cambiar de las 2D a las 3D, algunas de las funciones que usabamos con DIV ya no sirven como es el caso de collision().

Autor: Carlos González

Los principios de la animación aplicados a videojuegos

La animación de escenas y personajes es una parte muy importante de un videojuego y, a veces, se descuidan algunos detalles, y las animaciones "pinchan" en algunas partes. Y es que debemos tener clara la idea que animar no es "mover", sino más bien dar "vida".

Desde hace décadas, los estudios de animación de Walt Disney han investigado la manera de dotar a sus personajes de gracia, naturalidad y "vida". Los resultados de estos estudios han dado una serie de principios para la animación, que pueden aplicarse tanto a la animación tradicional, 2D, como a las técnicas 3D. De estos principios podemos destacar diez que veremos a continuación.

SQUASH AND STRETCH

Aplastar y Estirar: Es, sin duda, el primer principio que tenemos que dominar. Una característica que deben aportar los animadores a sus personajes es masa y flexibilidad. Gracias a estas dos características, cualquier personaje con "vida" se deformará en su movimiento, pero teniendo en cuenta que no cambiará su volumen. Un personaje que ha sufrido *squash* presentará su forma achatada (en horizontal), mientras que *stretch*, por el contrario, muestra una forma alargada (en vertical). Pongamos algunos ejemplos: imaginemos una pelotita de goma que queremos animar. Si únicamente movemos una esfera por la pantalla, no parecerá realmente una pelota botando. Tendremos que aplicar el concepto de *squash* cuando toque con el suelo, y *stretch* cuando salga impulsada hacia arriba. Para hacer esto hay que recordar que no debemos variar el volumen del objeto. Imaginemos ahora un canguro saltando. El *squash* ahora no deberá deformar al canguro; bastará con encoger sus patas traseras, bajar un poco la espalda y, para el impulso hacia arriba, estirar las patas y erguir la espalda. Imaginemos ahora una bala de un cañón. Cuando salga del cañón, para dar más efecto de movimiento, deberemos aplicar un *stretch* debido a la gran fuerza a la que se ve sometida la bala.

TIMING

Temporización: Es la fuerza que actúa detrás del movimiento; es decir, la rapidez con que transcurrirá la acción dependiendo de las características físicas de los personajes, que deberán aparentar tener peso y musculatura (o volumen si es un objeto). Éste es un punto crítico para que la animación parezca real. De esta forma, dos pelotas que inicialmente pueden parecer iguales, con el mismo volumen, si las animamos para que caigan desde una misma altura y le aplicamos distinto *timing*, parecerá una distinta de otra (caerá antes; parecerá más pesada).

ANTICIPATION

Anticipación: No es suficiente con poner una animación; debemos estar seguros que nuestra animación va a ser entendida por todo el que la vea. Ésta parte tiene que estar muy trabajada y se deberá dedicar bastante tiempo en conseguir una "anticipación" realmente buena. Para ello, disponemos del *story-board* (del cual hablaremos en próximos artículos).

Cuando animemos cualquier cosa, deberemos incluir acciones secundarias respecto de una acción principal, así lograremos un mayor realismo

Cuando tengamos el modelo de *story-board* elegido, elaboraremos la animación y se la enseñaremos a diferentes personas que no sepan en un principio nada sobre ella. Después, podrás preguntarle qué han visto en la animación y si la comprenden, la "anticipación" es correcta. Esto nunca deberá llevarte a hacer animaciones para niños, pero es muy importante preparar al espectador para la acción que va a venir. Pongamos un ejemplo: imaginemos que en una secuencia de animación queremos que el personaje principal piense gastarle una broma a otro (la broma consistirá en poner pegamento en un asiento). La anticipación de la acción podría ser desde ver cómo el personaje principal se acerca al tubo de pegamento, o bien cómo coge el tubo de pegamento, cómo echa el pegamento en la silla o una "anticipación" tan sutil como una simple mirada del personaje principal en primer plano a un tubo de pegamento en segundo plano. De todas estas posibilidades (y más que se podrían barajar) tendríamos que escoger la que más gracia tenga (combinado con que el espectador no tenga que preguntarse: "¿y que está ocurriendo ahora?"; simplemente que disfrute viendo nuestro trabajo).

STAGING

Las ideas deben presentarse de una forma clara y sin confusión al espectador. Las acciones se deben entender, las personalidades (muy importante en la caracterización de personajes y de la que hablaremos más adelante) deben estar claras y tienen que ser identificables, y las expresiones (faciales y movimientos corporales) ser reconocibles. El animador no deberá expresar más de una idea en el mismo tiempo, y utilizar siempre el plano que más convenga (también hablaremos de los tipos de planos próximamente en esta sección). En 3D, sobre todo, la utilización de diferentes tipos de luces y su buena orientación será decisiva.

EXAGGERATION

Exageración: Es uno de los principios más utilizados en todas las animaciones. Desde los clásicos dibujos animados, pasando por las modernas animaciones 3D, la exageración está presente, sobre todo, en secuencias cómicas. Tanto si la secuencia es cómica como si no, se deberá utilizar la exageración para dar más viveza, más fuerza a la animación.

Una característica que deben aportar los animadores a sus personajes es masa y flexibilidad

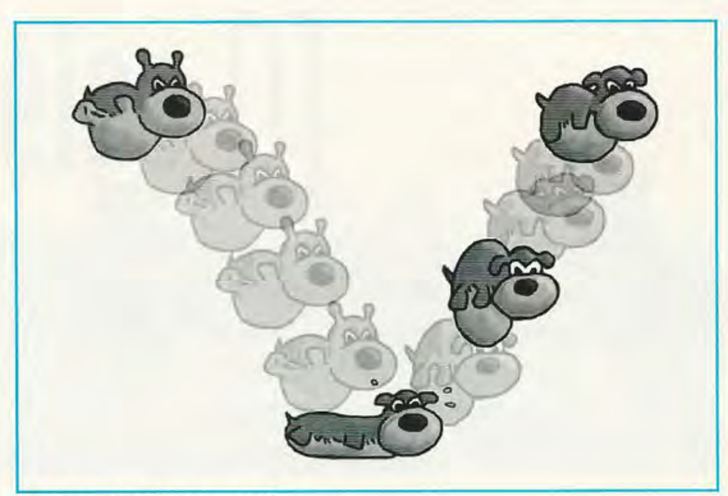
Pero el espectador no deberá pensar que está viendo algo surrealista (animación con demasiada exageración), sino que la exageración debe ayudar a expresar la idea con más facilidad, asegurándonos que el espectador comprenderá la acción. Deberemos extraer la esencia del movimiento, y acentuarlo para que se entienda mejor qué está ocurriendo. Por ejemplo, todos hemos visto las balas de los cañones en dibujos animados. Cuando éstas tocan el suelo explotan, sin embargo, en la realidad no ocurre así (se quedan incrustadas en el suelo). Otro ejemplo de exageración usado en casi todas las animaciones son las expresiones faciales de los personajes.

FOLLOW THOUGH AND OVERLAPPING

Los objetos que llevan un movimiento no se paran de golpe. Primero se para su parte "líder", y luego los elementos subordinados a esa parte líder. Ésta es la que realmente lleva el "peso" de la animación, y se moverá más rápidamente y tardará menos en pararse que las partes subordinadas. Por ejemplo: imaginemos una persona corriendo. Si queremos que detenga su carrera, los brazos, el pelo y otras partes subordinadas continuarán moviéndose y pararán de moverse más lentamente que el tronco (que consideraríamos como "parte líder"). Debemos tener en cuenta que para pasar de una acción a otra no hay que finalizar una y luego empezar con la otra, sino que podemos empezar una acción sin haber terminado completamente la anterior. Imaginemos a nuestro personaje en una batalla. Los enemigos le disparan por todas partes, ve un fusil y corre a por él. No quedaría bien que corriera hacia el fusil, parara su carrera, cogiera el fusil, después se tirara al suelo, apuntara con el fusil y disparara. En este caso lo más conveniente sería que el personaje corriera, al ver el fusil se tirara al suelo a por él (y en el suelo, terminara su carrera, sin pararse previamente), mientras coge el fusil, podría adaptar su posición para apuntar. Una vez parado, empezar a disparar.

SECONDARY ACTION

Acción secundaria: Es una acción subordinada a otra. Cuando animemos cualquier cosa, deberemos incluir acciones secundarias respecto de una acción principal. Con esto conseguiremos un mayor realismo. Por ejemplo: en una escena que sea un tiro de baloncesto, la acción principal será el tiro (movimiento de la mano, hombro, brazo, etc..). Si sólo animamos esto, la animación "canta". Deberíamos incluir acciones secundarias como el público que se levanta de sus asientos, la mirada del jugador se eleva, el cuello se estira, la cabeza asciende... Esto no debe llevarnos a contar más de una idea principal en el mismo tiempo, pues el espectador no sabría donde mirar. Más bien, debemos acompañar las acciones principales con algunas acciones secundarias (siempre que sea posible).



Una prueba de Squash y de Stretch.

SLOW IN AND SLOW OUT

Este principio habla de la distribución de los frames que forman una animación. Los keyframes claves son los puntos críticos de una animación. Imaginémonos nuestra típica pelotita botando. Un keyframe clave sería el punto más alto de su trayectoria, y otro keyframe clave, el choque contra el suelo. Imaginemos ahora una persona saludando. Un keyframe clave (kfc) sería la persona con la mano hacia un lado, y el otro kfc con la mano girada al lado opuesto. El espacio que hay entre dos kfc se denomina *inbetweens*. Los animadores de Disney se dieron cuenta que el espacio entre keyframes clave no tenía que ser constante. Si se altera este espacio y, por ejemplo, ponemos más frames entre los keyframes clave extremos de un movimiento rápido, conseguimos un movimiento fugaz. Si, por ejemplo, concentramos frames antes de un keyframe clave, conseguiremos dar la impresión de que el personaje se está parando... Imaginemos al correcaminos en la carretera; si queremos dar la sensación que empieza a correr y para más adelante, tendríamos que concentrar frames en el keyframe clave inicial y final, y dibujar pocos frames intermedios, así como acompañar a los frames intermedios con algunas líneas cinéticas para darle más impresión de velocidad.

Para saber más

Si estáis interesados en profundizar sobre este tema, os recomendamos, sobre todo, el libro de Walt Disney "Illusion of Life". Podéis buscarlo en grandes bibliotecas o pedirlo de encargo (ya que es difícil de encontrar), creemos que sólo está publicado en inglés, y su precio es algo elevado. De cualquier forma es la referencia bibliográfica básica que conocen todos los animadores profesionales; así pues es muy aconsejable. También os aconsejamos algunos artículos del Sigart Bulletin (Volumen 8 de 1997, que recoge los números del 1 al 4), como el escrito por Tom Porter y, especialmente, el artículo de John Lasseter "Principles of Traditional Animation Applied to Computer Animation" de la publicación Computer Graphics (21-24) del mes de julio de 1987. Ambos en inglés. En los primeros números de la publicación 3D World (de nuestra editorial), Daniel Martínez hace un vasto repaso de los principios de animación aplicados a las 3D.



Nuestro perrito os saluda.

ARCS

Si observamos la naturaleza, muy pocos movimientos van en línea recta. Lo natural son movimientos curvos (normalmente arcos). En los programas de animación por ordenador normalmente manejamos trayectorias rectas.

Una cosa que tiene appeal es una cosa que te gusta ver, que te sientes cómodo de ver

Pues bien, eso no debe hacerse, sino que debemos esforzarnos por hacer trayectorias más o menos curvas (según necesidad del movimiento) pero nunca (o en muy contadas ocasiones) trayectorias rectas, ya que disminuye la sensación de naturalidad del movimiento.

APPEAL

Su traducción al español es atractivo. Una cosa que tiene *appeal* es una cosa que te gusta ver, que te sientes cómodo de ver. Las animaciones pueden estar faltas de *appeal* cuando el dibujo es malo, o cuando hemos descuidado algunos aspectos. Algunos fallos de los más comunes son poner al personaje principal con los brazos y las piernas en la misma posición, que la cara sea simétrica respecto de la nariz, olvidar las articulaciones de los brazos y las piernas así como el balanceo del cuerpo el cualquier movimiento...

CONCLUSIONES

Estos 10 principios son aplicables a todas las animaciones que forman un juego. Si bien algunos principios (como *anticipation*) están destinados a ciertos tipos de juegos (como aventuras gráficas, y juegos con grandes secuencias de animación o animaciones a pantalla completa entre fase y fase), todos se deben tener en cuenta para las animaciones individuales de los personajes (conjunto de sprites) y demás elementos que formarán el juego. Teniendo en cuenta estos principios, nuestras animaciones ganarán muchísimo sin tener que mejorar nuestro nivel de dibujo, siempre que no olvidemos aplicarlos en todas las animaciones que hagamos.
Un elemento que muy pocas veces se tiene en cuenta a la hora de animar un personaje es dotarle de cierta "vida". Un personaje bien animado gana mucho, pero si no le definimos una personalidad y una forma de pensar, no parecerá que está vivo y, de esta forma, la persona que vaya a jugar a nuestro juego, no "sentirá" el personaje como si fuera real. Un ejemplo de buen hacer en este aspecto son todas las películas de Walt Disney, y para mencionar un juego pondremos como ejemplo la saga de los "Larry".

Estos 10 principios son aplicables a todas las animaciones que se incluyen en un videojuego

En cualquier grupo de desarrollo se deben hacer reuniones entre los miembros para discutir diferentes aspectos de la obra que van a crear. Uno de los que primero deberían tratarse es la personalidad de los protagonistas que intervengan en el juego. Al principio, puede parecer una pérdida de tiempo, pero es una parte muy importante y a la que se le deben dedicar bastantes reuniones. Como todos sabemos, no anda igual un tío muy chulo por un paseo con chicas guapas, que una persona deprimida u otro "cabreado". Si nuestro personaje es, por ejemplo, un tipo que se enfada por todo y le dan miedo las cucarachas, en el juego, estará enfadado, andará decisivamente por todos los escenarios, empujando a otros personajes, destrozando todo y si ve una cucaracha, se pondrá a temblar, se subirá a una silla...

Así, deberemos evitar que nuestros personajes queden sosos, sin "vida". Debemos dar a entender no simplemente que el muñequito se mueve, sino que ese conjunto de píxeles tiene un cerebro, que siente, que tiene su personalidad. Una muestra espléndida de esto podemos verla en Toy Story; los personajes Woody y Buzz tienen su personalidad, su forma de actuar ante diferentes situaciones y el público tiene una disposición distinta con un personaje y otro. Éste es el objetivo a conseguir por cualquier animador. Además, si dotamos a nuestro personaje de una serie de debilidades y defectos, conseguiremos mayor efecto de realismo (el ideal de fuerza, belleza, valentía en un personaje puede producir el efecto contrario al buscado; el espectador perderá interés ante el típico personaje que ha visto mil veces en muchos juegos). Debemos cuidar también la forma de contar la historia. En grandes secuencias de animación es IMPRESCINDIBLE la elaboración de un Story-Board entre todos los miembros del grupo de desarrollo (principalmente, diseñador, grafistas, animadores y músicos). De este aspecto, junto con una descripción de los tipos de planos y su utilización, hablaremos en el próximo número del suplemento Game Developer.

Este personaje posse un gran atractivo.



Autor: Juan Suárez

El diseño de niveles [I]

Un mes más proseguimos con la andadura de esta exclusiva sección, que os propone llegar a conocer, poco a poco, todos los entresijos del desarrollo de un videojuego, con un planteamiento unas veces más teórico y otras más práctico.

Hoy nos toca hablar del diseño de niveles. Éste es, quizás, uno de los procesos más enriquecedores y creativos en el desarrollo de un videojuego, ya que nos permite ver, prácticamente "in-situ", cómo va a ser nuestro juego. El diseñador de niveles tiene en sus manos todo el trabajo del resto de componentes del equipo desarrollador: cientos de rutinas, gráficos, sonidos, y motores poligonales 3D, de física, e Inteligencia Artificial, etc... Así pues, es a él a quien corresponde combinar esos elementos para dar vida al juego, y hacer divertida y creíble la experiencia de jugar. Probablemente esto es lo que hace tan enriquecedor el proceso de creación de un nivel, ya que el diseñador deberá conocer a la perfección todo el material de trabajo ajeno con que cuenta, así como sus limitaciones y posibilidades, para procurar sacarle el mayor partido posible. Pero vayamos por partes, porque... ¿qué es en realidad un diseñador de niveles?

EL DISEÑADOR DE NIVELES

Aunque suene a perogrullada, la respuesta obvia a la pregunta anterior, es que, el diseñador de niveles es aquel que diseña niveles... Vale, muy bien, pero..., ¿qué es entonces un nivel? Y..., ¿cómo se las apaña uno para diseñarlo...? Pues bien, un nivel no es otra cosa que un entorno simulado, compuesto de diversos elementos que, combinados correctamente entre sí, recrean un escenario que resulta de alguna forma familiar para el jugador, haciéndole sentirse identificado con él o en él, y que le supone a todas luces una experiencia, cuando menos, creíble, y en la medida de lo posible, divertida. Visto de una forma más sencilla, el diseño de un nivel implica que éste sea creíble, convincente; a fin de que el jugador se deje llevar por él, se sumerja en el nuevo mundo que le es presentado ante sus ojos, y consiga hacerle olvidar por unos instantes la realidad. Pero..., ¿qué es lo que hace entonces creíble un nivel? Sin duda alguna, hacer que un nivel sea creíble supone llegar a crear una realidad alternativa. Para ello, se necesita alcanzar unas cotas de detalle y concepción de diseño lo suficientemente elevadas como para que, una vez nos hayamos introducido en él, nos sintamos parte de un nuevo mundo en el que la lógica imperante, sea continua y consistente; una realidad virtual, paralela, incluso distinta...
Los diseñadores de niveles son los responsables de esta lógica, que no hay que confundir con la lógica interna que rige el programa. Esta lógica a la que nos referimos es la que hace que todo cuadre, que nos parezca que cada cosa está en su sitio, que nos sintamos identificados en el mundo que se nos presenta, porque éste nos resulta completamente coherente. Lo que puede hacer perder la credibilidad de un entorno es, por ejemplo, el desplazamiento o el mal "tileado" de las texturas que decoran nuestro escenario, y que suele ser un fallo habitual cometido en muchos niveles. Otro claro ejemplo de la pérdida de credibilidad puede ser un conjunto de puzzles que no tengan ningún sentido en una videoaventura; o un modelo de física inadecuado en un juego de carreras, que haga completamente irreal el comportamiento de los vehículos que se vean sometidos a dicha física. Cosas como éstas son las que hacen que el jugador, en un abrir y cerrar de ojos, pierda por completo la ilusión que tenía de estar soñando despierto. Que se vea transportado de nuevo a la realidad convencional, como si hubiese despertado de un sueño, haciéndole tomar conciencia una vez más de que, simplemente, estaba jugando al ordenador...
Queda claro, pues, que el diseñador debe, cuando menos, ser capaz de recrear un entorno que resulte creíble, donde el ideal sería aquel en que todos los elementos presentados estuvieran en sintonía con las percepciones del jugador. Así que, prosigamos con un poco más de teoría y conozcamos qué es lo que hace creíble una experiencia virtual.

PRINCIPIOS DE LA CREDIBILIDAD

Tres son los principios que definen la credibilidad de un juego:
1) El sentimiento de "*estar ahí*": Este principio contestaría a la pregunta de, ¿nos sentimos realmente integrados en el mundo imaginario que se nos presenta?, ¿creemos estar en una nave espacial, o en un castillo abandonado, a los mandos de un flamante bólido, o sobrevolando el Sáhara? Esto es lo que supone "sentirse ahí", esa sensación de que el protagonista de la acción es uno mismo, y que la lógica que rige el entorno nos es conocida y sabemos controlarla para conseguir nuestro objetivo en el juego, y todo ello sin perder esa separación temporal de la realidad. Si aún así no lo entiendes..., échate un Quake y verás a qué nos referimos...
2) La creación de emociones: ¿Nos sentimos realmente aterrados ante el monstruo que se nos avecina, o armados de valor, o cualquiera que sea la emoción que nos deba transmitir el juego en ese momento? En definitiva, nos referimos a la provocación de emociones en el jugador, haciéndole creer que ya forma parte del mundo que hemos preparado con tanto esmero para él... En este sentido, los apartados gráfico y sonoro tienen un rol fundamental, así como cualquier otro elemento que contribuya a aumentar la credibilidad. Porque, ¿quién no se ha llevado un susto de muerte jugando a Quake cuando, de repente, aparecía un feroz demonio de detrás de la compuerta que acabábamos de activar?, ¿o ha sentido en sus propias carnes el dolor producido por el crujir de huesos después de caer desde 20 metros de altura?, ¿o casi se asfixia bajo el agua, de la que acaba de salir con una necesidad indescriptible de oxígeno?
3) El factor "*diversión*": ¡Dios! ¡Cómo mola! ¡Has visto lo que hace....! Expresiones como estas son típicas en auténticos jugones. Gente que enseguida aprecia la cantidad de detalles que tiene un buen juego, y disfruta y se divierte con ellos. En común con el resto de jugadores..., el

PROFUNDIDAD DE LOS MAPAS = PROFUNDIDAD DE JUEGO.




La proyeccion de sombras aumenta el realismo.


Escaleras y objetos de decoracion dan vida al mapa.

hecho de que les permita hacer cosas que en la cruda realidad seguramente no podrían. La diversión, o jugabilidad para ser más correctos, se transmite como en ningún lado en el diseño de un nivel, siempre que la programación y los demás elementos lo apoyen. Recrear al jugador con trampas espectaculares, o animaciones sorprendentes e inesperadas, enemigos que supongan un auténtico reto y cosas por el estilo definen la jugabilidad de un juego. Y, aunque parezca una tarea trivial, supone muchas horas de juego reiterativo y retoque del entorno creado, hasta lograr el grado de *diversión* o *jugabilidad* deseado...
Por supuesto, estos principios son papel mojado si no se ven apoyados por un buen "frame-rate", una programación potente, o unos gráficos decentes, que permitan a la mano sabia y creadora de un buen diseñador de niveles, extender sus dotes y virtudes de "jugón", para recrear e impresionar y, sobre todo, divertir a todo aquel que se atreva a probar...

EL OBJETIVO A CONSEGUIR...

Se podría decir, en términos *"pofesionales"*..., que un nivel es aquel entorno en el que todos los demás elementos del desarrollo -programación, gráficos, sonidos, etc...- se ven representados. Así que es al diseñador de niveles a quien corresponde hacer quedar bien al resto de componentes del equipo. Él es quien tiene todos los elementos necesarios para hacer del juego una experiencia y, eso amigos..., eso es una experiencia en sí... A veces nos sentiremos frustrados por no poder hacer cierto tipo de cosas, como emplear más texturas para decorar nuestro escenario, o emplear más objetos que ayuden a ambientar mejor el entorno,... Pero el poder de creación que tenemos en nuestras manos es absoluto. Y eso..., como decíamos antes, es una experiencia también... Pero..., no perdamos los papeles todavía... Tener ese privilegio exige que uno conozca casi a la perfección todo lo que tiene a su disposición. Desde las limitaciones de memoria para texturas, número de polígonos, entidades lógicas, o animaciones; hasta las incapacidades con que pueda contar la herramienta que vayamos a emplear para la creación de niveles.

Esto, implica que el diseñador deberá saber relacionarse con el resto de componentes del equipo desarrollador, es decir: grafistas, programadores y músicos. La fluidez de ideas entre los miembros de un equipo desarrollador es vital, y también muy gratificante. Compartir ideas y explotarlas más allá de lo que nuestra propia cabeza da de sí, acaba no sólo por crear un buen ambiente de trabajo sino, además, una compenetración absoluta entre los componentes del equipo que, más tarde o más temprano, acaba por dar lugar a grandes creaciones. En la medida de lo posible, el diseñador de niveles debe saber cuanto pueda sobre la clase de rutinas empleadas en el juego, tanto a nivel de *"engine"* 3D como de lógica o física. En definitiva, deberá procurar conocer todos los conceptos teóricos que rigen dichas rutinas, para llegar así a comprender qué es lo que le va a permitir hacer que el juego sea vistoso y divertido. Y esto sólo se consigue aprendiendo de los demás, y desarrollando y ampliando nuestros conocimientos por cuenta propia,... así que el ser gente espabilada y emprendedora nos será de gran ayuda. Cada vez más, se exige de un diseñador que sea capaz de "ver el cuadro" en su conjunto, es decir, ser capaz de abstraer el concepto del propio juego y plasmarlo de múltiples formas en un nivel. Antiguamente, los diseñadores de niveles se centraban en hacer mapas o escenarios que fuesen capaces de valer por sí solos, como si cada nivel fuese un juego dentro del juego. En la actualidad, el diseño de niveles se está empezando a ver íntimamente ligado al propio diseño y concepción del juego, y a su coordinación. Pero, ¡quieto "parao"...! Para llegar a eso hay que empezar por el principio, porque..., llegar a "ver el cuadro" no es cosa de dos días. Así que..., Daniel Sam...deber primero aprender que..., toj, toj, ...lugar no ocupa, saber en su cabeza..., porque..., toj, toj, ...en conocer las cosas todas..., estar sabiduría... toj,...toj..., toj...

EL CONOCIMIENTO ES LA CLAVE

Cual maestro *Jedi* os digo..., que la fuerza está en vosotros... Hay que llegar a aprender muchas cosas, tantas como podamos. Y eso exige, si no cierta disciplina sí mucha ilusión, empeño y sacrificio, o ambas cosas. Cualquiera que se quiera tomar en serio el diseño de niveles y, además, poder vivir bien de ello, tendrá que obviamente familiarizarse primero con un ordenador, lo cual no es mucho pedir...
El curriculum de un diseñador de niveles tiene difícil descripción, pero lo que está claro es que hay una serie de aspectos que facilitan y potencian enormemente el proceso de diseño, con los que todo diseñador debería sentirse familiarizado. Para empezar, se requiere haber probado y probado juegos hasta la saciedad..., analizando qué es lo que los hace divertidos, qué es lo que más impresiona de ellos, pequeños detalles que supongan grandes alicientes, etc... Creemos obvio que, en orden para saber hacer divertido un juego, se necesita conocer primero qué es lo que hace divertidos a otros juegos. Después, deberíamos aprender a manejar alguna herramienta de creación tridimensional, como 3D Studio o LightWave, o bien algún mapeador de dominio público, como Worldcraft. Sólo con eso ya podremos iniciar la creación de nuestros propios diseños. Lo más fácil es crear estructuras conocidas y sencillas, como nuestra casa o nuestro barrio, para luego pasar a imitar diseños de otros juegos, y finalmente construir aquellos de los que nos podamos sentir enormemente orgullosos, donde habremos puesto toda nuestra sabiduría y saber hacer... Pero antes de llegar a ser capaces de "crear" por nosotros mismos diseños válidos y fuera de lo común, necesitaremos un conocimiento muy genérico y amplio, que es sólo conseguido mediante la observación y el aprendizaje evolutivo. Y que necesita de un fuerte apoyo en cuestiones que nada tienen que ver, aparentemente, como el arte o la estética, o incluso el sentido práctico. Pero..., ¿cómo puedo llegar a aprender todo eso? Pues, *"a los fechos me repito..."*, que diría el Pazos...

QUE APRENDER Y COMO...

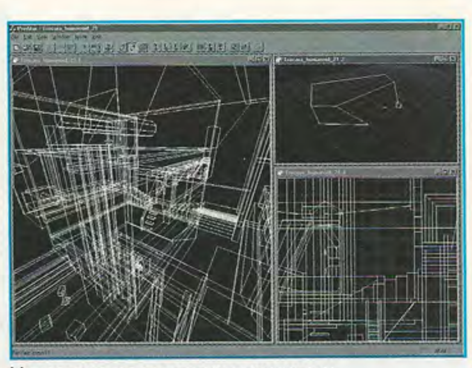
Aunque no se requiera de una educación especial, la propia enseñanza que se nos imparte en las escuelas, institutos y, posteriormente, en la universidad nos provee de un sólido cimiento sobre el que asentar nuestros conocimientos sobre diseño de niveles. Cosas tan básicas y elementales como el saber algo de arte contemporáneo o historia, filosofía, y, mejor aún, literatura, ayudan enormemente. Cuantas más experiencias tengamos en nuestro haber, como haber viajado, conocer otras culturas, tradiciones, distintos estilos arquitectónicos, etc..., más y mejores serán las armas con que contaremos a la hora de diseñar nuestros niveles. Todo ello nos ayudará a encontrar nuevas ideas que plasmar, nuevos conceptos que explotar y toda una experiencia que compartir. Como podréis observar, y tal y como os habíamos adelantado, éstas no son cualidades que se puedan aprender en una escuela concreta, o en un determinado



master o postgraduado de alguna facultad.
La adquisición de dichas cualidades requieren del esfuerzo, por parte de uno, de querer saber más, aprender más, y saber explotar el conocimiento adquirido para ponerlo al servicio del diseño.
La gran ventaja que esto tiene es que, a diferencia de un estudio, digamos forzado, como pueda resultar el tener que ir a clase todos los días, el individuo absorbe y asimila con mucha más facilidad toda la información que él, voluntariamente, ha decidido aprender. Algo que puede contribuir a desvelaros nuevas ideas con cierta facilidad es el ir a una tienda de libros, preferiblemente unos grandes almacenes, donde nadie nos venga a incordiar sobre si queremos o dejamos de querer algo... Teniendo más o menos clara cuál es la clase de orientación artística y ambiental que queremos dar a nuestro diseño, nos resultará sumamente fácil encontrar material de trabajo en multitud de libros, llenos de información e imágenes que, en ocasiones, nos podrán llegar a resultar reveladoras. De esos libros podremos no sólo sacar ideas para nuestros diseños, sino texturas, fondos y muchas otras cosas que posiblemente enriquezcan el nivel que nos proponemos diseñar y, en definitiva, el juego que nos proponemos hacer. Otro sitio, muy "en boga" estos días, donde encontrar información, ideas, y cualquier cosa de utilidad, es, cómo no..., Internet. Hoy por hoy existe mucha y variada información sobre prácticamente cualquier cosa que podamos imaginar. Pero, mejor aún, existe información concreta sobre el desarrollo de videojuegos y el diseño de niveles, entre otros, claro... Así que tendremos a nuestro alcance todo lo necesario para aprender más cosas, e incluso compartir ideas, porque son ya habituales los canales de "chateo" en el IRC, en los que se habla del desarrollo de videojuegos. Quizás podamos llegar a tener la suerte de hablar con algún componente, de los ahora numerosos grupos de desarrollo existentes en nuestro país. Seguro que ellos estarán encantados de resolvernos todas las dudas que les planteemos... Pero, esto es sólo la punta del iceberg..., porque, qué sitio si no Internet para conseguir las imágenes de los juegos más novedosos, o las demos jugables de los mejores videojuegos, mapas y niveles diseñados por otras personas, y un largo etcétera de cosas que contribuirán a nuestra formación como diseñadores. Y, además, qué mejor lugar para dar a conocer nuestras creaciones que Internet también. Una vez que te sientas seguro y conforme con tu diseño, intenta enseñárselo a alguien, procurando conseguir siempre todo el *"feedback"* que puedas, es decir, toda la información que esa persona te pueda dar sobre lo que le ha parecido tu nivel. Si le ha resultado divertido y por qué, si le ha parecido "cañero" y bien estructurado, etc... Nada como la observación sincera e imparcial de un extraño, para saber que nuestro diseño es un fracaso o, por contra, un excelente nivel en el que emprender batallas encarnizadas en modo *"death-match"*...

EN UN MES, MAS...

Bueno, queridos amigos, hemos terminado por hoy... En la próxima visita seguiremos descubriendo algún que otro entresijo más del apasionante mundo del diseño y el desarrollo de un videojuego... Conoceremos más sobre las herramientas empleadas en el diseño de niveles, que nos permitirán aprovechar el material gráfico y la programación desarrollada para ir haciendo nuestros pinitos. También descubriremos algunos consejos prácticos, que hagan de vuestros niveles algo no sólo divertido e impresionante, sino único y distintivo de vuestra personalidad... N'xaludito...y hasta el mes que viene...



UN EJEMPLO DE UN EDITOR PROFESIONAL.

FORO DE OPINION Game Developer

El suplemento Game Developer extiende sus fronteras. Y qué mejor manera de llegar a cualquier punto del mundo que haciendo uso de los servicios que pone a nuestra disposición la red de redes, Internet. Dentro de muy poco tiempo todos los seguidores del Game Developer podrán disfrutar de un nuevo canal IRC, dedicado al desarrollo de videojuegos. Si quieres formar parte de este apasionante y atractivo mundo, y nunca te han dado aportunidad de hacerlo, ésta es la tuya.

Esta lista es sólo un ejemplo de las posibilidades que te proponemos en él:

PROXIMO CANAL IRC DE GAME DEVELOPER

DESARROLLO DE VIDEOJUEGOS

- Charla con los responsables de los equipos de desarrollo.
- Conoce los secretos de los programadores de videojuegos.
- Opina sobre las compañías y sus políticas de desarrollo.
- Profetiza sobre las nuevas técnicas de programación.
- Discute sobre las mejores herramientas.
- Elige los mejores grupos de desarrollo, juegos, personajes, ...
- Trata temas de actualidad:
- Resuelve tus dudas entre los asistentes al canal.
- ...

Desde aquí os invitamos a que nos inundéis con cartas, faxes, e-mail, y demás con todas vuestras sugerencias, ideas, ocurrencias, y opiniones sobre el tema que puedan ayudar a la hora de desarrollar este nuevo medio de debate.

Manda tus sugerencias a la siguiente dirección:

GAME DEVELOPER

c/ Alfonso Gómez, 42, NAVE 1-1-2,
CP 28037 Madrid

E-mail: gover@prensatecnica.com
fax: (91) 304 17 97

GAME DEVELOPER SIEMPRE AL SERVICIO DEL DESARROLLO Y LOS DESARROLLADORES

Autor: Víctor Segura

Cake Walk [I]

Durante este mes y los siguientes, analizaremos uno de los mejores secuenciadores MIDI del mercado: el Cakewalk. Mostraremos los diferentes métodos a seguir para componer nuestra banda sonora y como no, comentaremos algunos "pequeños trucos", que serán bastante útiles a la hora de desarrollar nuestra propia música.

El porqué del Cakewalk y no otro secuenciador es simplemente porque es uno de los secuenciadores mejor asentados (al menos en el mercado español), y porque a medida que va pasando el tiempo es el que mejor ha evolcionado en la comercialización de nuevas versiones. De todas formas, si no eres un usuario de Cakewalk, y utilizas otros secuenciadores como pueden ser el Logic, el Orchestator, Cubase etc.. no te preocupes, ya que por lo general la única diferencia que existe entre unos secuenciadores y otros es el interface gráfico, y que a la hora de acceder a una función en concreto en un secuenciador, se hace pinchando en el boton derecho del ratón y accediendo al menu deseado, mientras que en otro secuenciador hay que acceder desde la barra de menús. Pero las posibilidades que ofrecen los diferentes secuenciadores son las mismas, y a medida que va pasando el tiempo tienden a estandarizarse de manera que la diferencia gráfica y la diferente manera de realizar las operaciones llegará a ser igual, prácticamente, tanto en un secuenciador como en otro.

¿Y LA VERSION 6.0?

La versión que vamos a comentar del Cakewalk no va a ser la 6.0. Lo que pretendemos es que se entienda y domine la base de los secuenciadores midi, y así cada uno podrá enfrentarse a los nuevos secuenciadores, pero sabiendo de antemano para qué sirve cada opción que nos ofrece el secuenciador y cómo se utiliza para sacarle el máximo rendimiento. Realmente, la diferencia más notable existente entre la versión 6 y las anteriores del Cakewalk es la incorporación de Audio digital y un entorno gráfico más espectacular. Pero como siempre decimos, más vale sacarle el máximo rendimiento a una versíon anterior (siempre que esta satisfaga nuestras necesidades) que utilizar la última versión si no nos ofrece mejores prestaciones y cada vez que queramos hacer algo nos perdamos en los menús, o no encontremos los controladores etc... Pero como hemos dicho anteriormente si se sabe manejar las versiones anteriores (la 4 o la 5), no nos encontraremos con ningún problema a la hora de manejar las más recientes.

UN PRIMER CONTACTO

Una vez cargado el Cakewalk, Lo primero que tenemos que hacer es cargar cualquier canción en formato MIDI o WRK (que es el propio del cakewalk) que tengamos.
Como en todos los programas vemos que tenemos un sistema de menus desplegables (por los que podremos acceder a las diferentes utilidades y funciones) y una ventana principal sobre la que realizaremos la mayor parte de nuestro trabajo.
Como podéis ver, en la ventana principal se pueden diferenciar dos ventanas: Una, la de Pista/Compas (Track/Measure) y la otra, la de control.

BARRA DE CONTROL

En la barra de control es donde accedemos a los diferentes controles globales de la canción (así como el tiempo, el compás, grabar, velocidad etc...).
Si nos fijamos en el dibujo 1 (dibujo 1.Barra de Control) vemos que hemos dividido en 5 las diferentes partes de las que se compone la barra de control. A continuación pasamos a describir cada una de ellas:

1. Controles Generales. Aquí tenemos los diferentes controles para movernos a lo largo de la canción (en concreto para situarnos en cualquier punto), para escuchar la canción y para grabar. Conocidos son los tres botones (REW, PLAY y REC) que aparecen en este punto. Sobre estos tres botones hay una barra de desplazamiento, que nos permitirá movernos hacia adelante o hacia atrás compás a compás o deslizando la barra hasta la posición deseada de la canción.

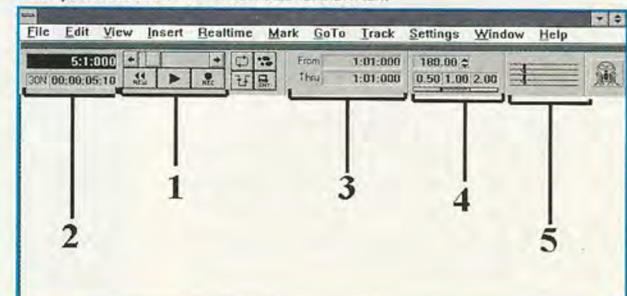
2. Tiempo. Aquí es donde tenemos la información de lo que dura nuestra canción así como el compás por el que va la canción. Estos controles están relacionados con la barra deslizadora anterior, ya que podemos deslizarnos con dicha barra y ver el tiempo de la canción en un punto exacto o el compás por el que vamos.

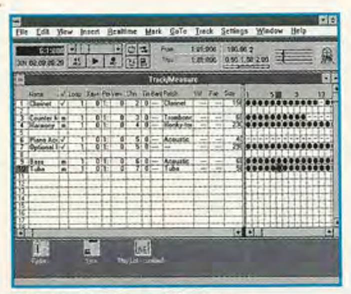
3. Selección. Con From-Thru podemos controlar dos puntos diferentes seleccionados por el usuario que luego pueden servir para utilizar funciones de edición (copiar, cortar, pegar) o para crear bucles. Estos controles son muy importartes a la hora de hacer las operaciónes que acabamos de describir, pero como veremos más adelante, esto se puede hacer sobre la propia partitura de nuestra canción. De hecho, la linea de trabajo a seguir en este apartado es seleccionar en la partitura y luego "afinar" con los controles From-Thru la selección.

4. Bmp. En el punto 4 obtenemos la información de la velocidad de la canción. En la parte superior tenemos la información de los Bmp (muy importante si luego que remos meter por ejemplo loops de baterias sampleados a una determinada velocidad), y en la parte inferior tenemos tres medidas: 0.50, 1.00, 2.00 medidas que si se seleccionan, y luego damos al PLAY, harán que la canción vaya a un 50, 100, 200 % de velocidad respectivamente. Esto es útil para meter melodías rápidas (así como solos). Si uno no dispone de la técnica necesarea para hacerlo tan rápido como quiere al piano, entonces pincha en 50% mientras la canción va a la mitad de velocidad, probablemente ya podrá introducir el solo a esa velocidad.

5. Compás. En este punto es donde recibimos la información del compás en el que estamos durante la canción. Así, si en un determinado momento tenemos un cambo de 2/4 a 3/8, ese

Dibujo1.Barra de control del Cakewalk.




DIBUJO 2. TRACK MEASURE.

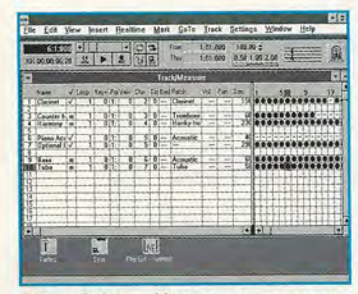
cambio se verá reflejado en esta parte de la barra de control. Como véis, tenemos en un principio un control absoluto sobre los diferentes controles globales de la canción.

VENTANA TRACK/MEASURE

Una vez explicada la barra de controles pasemos a la ventana más importante: La ventana Track/Measure. Aquí es donde vamos a realizar la mayor parte de nuestro trabajo: asignar instrumentos a pistas, tareas de edición sobre las diferentes pistas, control de los volúmenes y de los diferentes controladores, y prácticamente todas las operaciones que hagamos hasta que tengamos terminada nuestra canción.
Esta ventana esta dividida en dos partes: La relacionada con la pista (parte izquierda de la ventana) y la relacionada con los compases y su distribución en la canción (parte derecha de la ventana).
En la parte relacionada con las pistas podemos ver una pantalla dividida en diferentes celdas (como si fuera una hoja de cálculo), en las que cada columna significa una cosa diferente.

TRACK

La primera columna tan sólo informa del número de fila en que nos encontramos.
La segunda columna indica si el instrumento de esta fila (el nº 1 por ejemplo) esta en modo mute o no (si dicho instrumento permanece en silencio o no). Si lo que aparece es una "m" es que efectivamente está en silencio, con lo que no se oirá durante la ejecución de la canción. Si en caso contrario aparece un signo de "tick", quiere decir que dicho instrumento se oirá durante la canción. Esta opción es


DIBUJO 3. MENU HERRAMIENTAS DE EDICION.

muy útil para poder oir solo la batería y las guitarras sin el bajo (por ejemplo) y así ecualizar dichos instrumentos antes de realizar la mezcla final con el bajo.
Las siguientes columnas indican el nombre, el tamaño, el sonido asignado a la pista, el canal, el volumen y el balance de cada pista. Como véis en esta parte accedemos a toda la información de los instrumentos y sus características.

MEASURE

Lo que nos encontramos en la parte derecha de esta ventana no es más que un panel en el que se utilizan unos puntos para mostrar los compases en los que se ha grabado algún evento.
Vemos que esta parte es similar a una hoja cuadriculada. Pues bien cada cuadrícula indica un compás (cada fila será los compases del instrumento que este en la parte izquierda de la pantalla y en esa misma fila).
En la parte superior vemos una numeración (similar a una recla) con los números 1, 5, 9 etc..
esto nos sirve como referencia para seber el número de compás en el que estamos. Sobre estos números aparece un recuadro que va avanzando a medida que avanza la canción. Este recuadro indica el compas en el que nos encontramos de la canción.
Si queremos acceder a Track o a Measure, con hacer click en una de las dos pantallas de la ventana accederemos a ella. Con los cursores nos movemos por las diferentes pistas (en el caso de estar en track) o por los diferentes compases de cada pista (measure).
Herramientas de edición:
Si nos encontramos en la ventana Track/Measure, y hacemos click con el botón derecho del ratón (tenemos que mantener el botón clickeado), aparecerá (sobre el puntero del ratón) un menu con diferentes opciones relacionadas con la pista en la que hemos pinchado (recordad que la pista depende de la linea). Estas opciones las utilizaremos tanto en la elaboración de nuestra canción como en los retoques finales. De entre las 12 opciones vamos a tratar 4 que son realmente importantes.
1. Controllers
Con esta opción accedemos a los diferentes controladores MIDI. Aparecerá una ventana (vease dibujo3. Controladores) dividida en cuatro zonas.
Zona1: Nombre del controlador. En la zona 1 aparecerá el nombre (y número) del controlador que vamos a editar, y el canal del que estamos editando dicho controlador. es decir si accedemos al canal 2 que es donde tenemos un clarinete, los controladores que editemos serán solo los de este instrumento, y no afectará para nada al resto de los instrumentos y sus respectivos controladores.

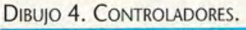
Cakewalk Application Lenguage

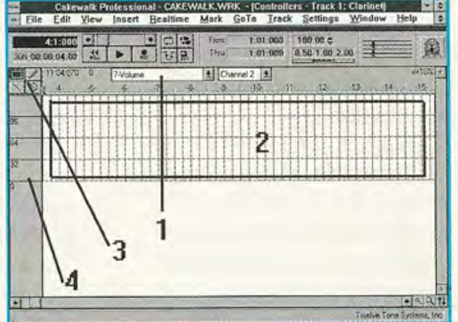
Una de las opciones más interesantes que nos ofrece el Cakewalk es la inclusión de un propio lenguaje de programación denominado CAKEWALK APPLICATION LENGUAGE (CAL) que traducido al castellano significa Lenguaje de aplicación de Cakewalk con el que podemos crear comandos informatizados. De hecho lo que realmente incorpora el Cakewalk es un lenguaje imperativo de programación con el que podemos realizar una serie de macros. Podríamos llegar a programarnos bases rítmicas, intercambios modales para sustituir una determinada melodía por otra y todo lo que podamos llegar a imaginar que se pueda programar dentro de la música. Aquí teneis una muestra de como programar un CAL program que dado una nota de al partitura, Ponga el acorde dominante séptima de dicha nota.

```
;; CH_DOM7.CAL
;;
;; Treats each note as the root of a dominant 7th chord; creates that chord:
;; For each note event, adds three note events with same time, vel, and dur
;; but a major 3rd, a perfect 5th, and a minor 7th higher.

(do
  (include "need20.cal") ; Require version 2.0 or higher of CAL
    (forEachEvent
        (if (== Event.Kind NOTE)
          (do
            (insert Event.Time Event.Chan NOTE (+ Note.Key  4) Note.Vel Note.Dur)
            (insert Event.Time Event.Chan NOTE (+ Note.Key  7) Note.Vel Note.Dur)
            (insert Event.Time Event.Chan NOTE (+ Note.Key 10) Note.Vel Note.Dur)
          )
        )
    )
)
```

Para acceder a esta utilidad lo haremos mediante la barra de menus con: View>Cal


DIBUJO 4. CONTROLADORES.



Archivo nuevo

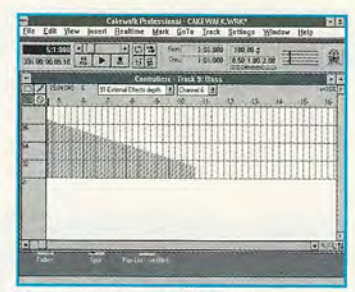
Cada vez que damos a la opcion FILE>NEW, nos saldra una caja de diálogo preguntando que tipo de archivo nuevo quiero: Normal (es decir en blanco), JazzTrio (aparecerá la Track Window preparada con los instrumentos correspondientes a este estilo)etc...
Los diferentes estilos son: 16track, bigband, brasquin, jazzquar, jazztrio, motu_min7,multimed, normal, orchestra, otto1604, piantrio, rockquar, strquar, woodquin.
Aparte de la plantilla instrumental, incorpora un texto en el que indica los rangos de los diferentes instrumentos así como su afinación. Aquí tenéis un ejemplo del Orchestra:
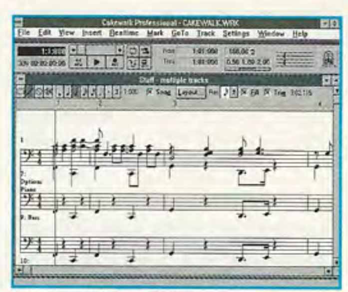
(Middle C is C5 in Cakewalk)

Instrument	Key+	Pan	Range (untransposed)
Piccolo	12	56	D6 to C9
Flutes	0	60	C5 to C8
Oboes	0	68	Bb4 to G#7
English Horn	-7	72	E4 to A6
Bb Clarinets	-2	58	D4 to Ab7
A Clarinets	-3	58	C#4 to G7
Bass Clarinet	-12	56	Db3 to F6
Bassoon	0	70	Bb2 to Eb6
Contrbassoon	-12	72	Bb1 to Eb4
French Horns	-7	64	D3 to F6
Trumpets	-2	70	E4 to E7
Trombones	0	80	E3 to Bb5
Tuba	0	92	A2 to Bb4
Timpani	0	50	n/a
Bass Drum	0	48	n/a
Cymbal	0	40	n/a
Triangle	0	40	n/a
Celesta	0	38	C5 to C9
Piano	0	28	A2 to C9
Harp	0	18	Cb1 to G#8
Violins	0	15	G4 to B7
Violas	0	88	C4 to C#7
Cellos	0	105	C3 to C6

Zona2: Area de Edición. En está zona es donde se cambian los valores de los controladores (por ejemplo del reverb, del corus, del sustain etc...) en los diferentes compases (indicados en la parte superior de esta zona con su número correspondiente)
Zona3: Herramientas de trabajo. En esta zona tenemos las cuatro herramientas (seleccionar, pintar, borrar y pintar linea) con las que modificaremos los controladores.
Zona4: Valor del controlador. A mayor altura mayor valor del controlador.
En el dibujo 5, tenéis un ejemplo de un fade de volumen desde el compás 5 al 10 utilizando la herramienta de pintar linea. El número del controlador es el 7.
Si queréis más información acerca de los controladores MIDI en el número 3 de Game Over podéis encontrar esta información y mucha más acerca del MIDI en general.
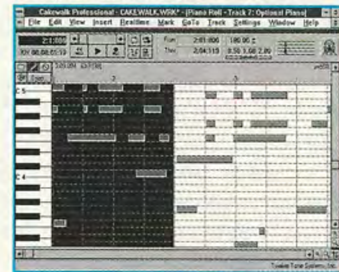
2. Staff
Cuando accedemos a esta opción (click botonderecho> Staff) aparecerá una ventana con la partitura de las pistas seleccionadas. En esta opción podemos retocar la canción instrumento a instrumento y nota a nota. Esto nos permite un control total y absoluto sobre nuestra canción. Por ejemplo si en la grabación de una melodía nos hemos equivocado en una nota, no tenemos que parar, repetir la grabación, y si sale bien guardarla, si no que utilizamos esta opción y cambiamos la nota erronea por la correcta.
También podemos editar una melodía nota a nota (recurso muy utilizado en la elaboración de las baterías), para conseguir así la mayor precisión posible. Pero os recuerdo que, por ejemplo, si metéis un bajo, nota a nota, perderá el feeling del directo, todas las notas se tocarán en su tiempo justo con el mismo volumen y con la misma intensidad. Sin embargo, si lo tocáis directamente desde un bajo o un teclado, el feeling será mayor.
Tenemos las opciones de seleccionar, insertar nota, oir nota y borrar. Si queremos modificar una nota, basta con pinchar con el botón izquierdo sobre dicha nota y (manteniendo el botón apretado) arrastrarla hasta el lugar deseado del pentagrama.


Dibujo 5. Fade de Volumen.


Dibujo 6. Staff Window.

3. Piano Roll
Del mismo modo que accedíamos a las opciones anteriores, podemos acceder al Piano Roll, (Rollo de Pianola) ideal para aquellas personas que carezcan de conocimientos de lectura musical y quieran retocar duraciones de notas y su intensidad en vez de en las partituras (staff windows) en un entorno más sencillo e intuitivo.
En resumen, que permite editar los eventos MIDI en un formato gráfico.

4. Event List
La cuarta herramienta de edición (y no por el hecho de ser la última la menos importante) es la lista de eventos MIDI, que permite visualizar y editar cada evento MIDI en la grabación.

Dibujo 7. Piano Roll Window.


En el dibujo 8 vemos que esta ventana está dividida en 6 columnas: Pista, SMPTE, compas, canal MTDT, tipo de evento y valores.
Esta herramienta es muy util para poner por ejemplo todas las notas de una melodía con la misma intensidad. Accedemos (como si fuera una hoja de cálculo) a la última columna (valores) y cambiamos el valor (situado en el centro) por el deseado, así con todas las notas hasta que hayamos terminado.

CONCLUSION

Como véis podemos llegar a controlar nuestra canción y realizar así canciones en las que podamos expresar al máximo detalle lo que queremos decir con nuestra música. Herramientas como el cakewalk con sus evoluciones, permiten que podamos llegar hasta unos límites (dentro del campo musical) que sin la ayuda de la informática serían inalcanzables. El próximo mes seguiremos analizando este secuenciador para sacarle el máximo partido. Pensad que si vuestro objetivo es ser unos "pequeños Vangelis", con un secuenciador como el cakewalk y unos buenos sonidos en un módulo o una tarjeta, podréis llegar a conseguirlo con un poco de esfuerzo.

Dibujo 8. Eventlist Window.